

# 2020학년도 2학기 동아리 수업 고난도 문제 풀이 모음

선린인터넷고등학교 소프트웨어과  
30610 나정희  
<https://JusticeHui.github.io>

## Abstract

2020년 2학기 선린인터넷고등학교 알고리즘 연구반 PS 수업 자료입니다.  
[solved.ac](https://solved.ac) 기준 Platinum 3 ~ Diamond 1 정도 난이도의 문제를 다룹니다.

여기에 수록되어 있는 문제는 <http://icpc.me/w/6132>에서 볼 수 있습니다.

전체 수업 자료는 <https://github.com/justiceHui/Sunrin-SHARC>에서 볼 수 있습니다.

## 0 문제 목록

1. 4 XOR MST - Educational Codeforces Round 32 G번	03
2. 3 Ploughing - POI 2005/2006 Stage2 6번	04
3. 3 Type Printer - IOI 2008 Day1 1번	05
4. 5 금광 - KOI 2014 중등부 4번	06
5. 3 장애물 경기 - KOI 2016 초등부 4번	07
6. 1 Favorite Colors - USACO 2020 US Open Gold 2번	08
7. 1 Springboards - USACO 2020 January Gold 3번	09
8. 2 Wall Clocks - ICPC Tsukuba Regional 2015 D번	10
9. 3 최단 경로와 큐리 - Good Bye, BOJ 2019 G번	11
10. 3 케이크 3 - JOISC 2018/2019 Day4 1번	12
11. 1 벽 칠하기 - APIO 2020 1번	13
12. 4 Xtreme NP-Hard Problem - 2018 KAIST MOCK X번	14
13. 2 Remote Control - 2020 KAIST MOCK I번	15
14. 5 성벽 - JOI 2015 5번	16
15. 4 NP-Hard - COCI 2013/2014 Contest2 4번	17
16. 4 Namje Adventure - 2018 아주대 교내 대회 Div1 G번	18
17. 2 타일 놓기 - TopCoder SRM 383 Div1 Med번	19
18. 3 데이터 제작 - UCPC 2020 K번	20
19. 2 부분 문자열의 개수	22
20. 5 Mountains - IOI 2017 Day0 1번	23
21. 3 소방차 - KOI 2005 고등부 3번	24
22. 4 편식 - COI 2015 4번	25
23. 1 Stretch Rope - 2017 Google APAC RoundD D번	26
24. 1 Alternating Current - Baltic 2018 Day2 A번	27
25. 2 야바위 - 2020 USIST 교내 대회 G번	29
26. 5 Confuzzle - 2020 서강대 교내 대회 Div1 G번	30
27. 1 모래시계 2 - 2020 연세대 교내 대회 J번	31
28. 1 Salty Fish - Petrozavodsk Camp Summer 2019 Day1 A번	32
29. 1 우체국 3	33
30. 3 3D Histogram - COCI 2020/2021 Contest1 3번	34
31. 1 겹치는 선분 - TUD Contest 2005 3번	36
32. 1 Synchronization - JOIOC 2013 2번	37

## 1 XOR MST

각 가중치는 29bit로 표현할 수 있다.

xor로 표현되는 가중치를 최소화하는 것이 목적이기 때문에, Binary Trie를 생각해볼 수 있다.

Boruvka Algorithm과 유사한 방식으로 MST를 구축할 것이다.

29번째 비트가 켜져 있는 정점 집합과 꺼져 있는 정점 집합을 있는 간선은 **한 개**만 있는 것이 최적이다. 켜져 있는 원소들로 Binary Trie를 구축한 뒤, 꺼져 있는 원소와 XOR 했을 때 최소가 되는 값을 찾아서 연결하자.

29번째 비트가 다른 원소 집합이 서로 연결되었다.

이제, 두 집합에 대해 각각 처리해줘도 된다. 두 집합에서 각각 28번째 비트에 대해 똑같은 과정을 처리하고, 27, 26, ...을 반복하면 문제를 풀 수 있다.

<http://boj.kr/e42aa1312ba342fa85bfdb21713f3161>

## 2 Ploughing

열심히 직사각형을 깍아내다 보면 마지막에는 가로 혹은 세로 길이가 1인 직사각형을 없애게 된다.

그 직사각형의 크기를  $1 \times K$ 라면, 직사각형을 총  $N + M - K - 1$ 번 깎아내게 된다.  $N + M - 1$ 은 일정하므로  $K$ 를 최대화하면 된다.

마지막에 가로로 길쭉한 직사각형을 남긴다고 하자.

마지막까지 남겨놓을 직사각형의 양 끝 점을 고정하면, 그런 직사각형을 남겨놓을 수 있는지  $O(N + M)$ 에 판별할 수 있다. 총  $O(M^2)$ 가지의 경우가 있기 때문에 시간 복잡도는 세제곱 수준이다. 최적화를 해보자.

양 끝 지점을 투포인터로 관리할 수 있다. 그러면  $O(M)$ 가지의 경우만 고려하면 되고, 제곱 시간에 문제를 풀 수 있다.

세로로 길쭉한 직사각형을 남기는 경우는 직사각형을 회전해서 똑같이 처리하면 된다.

<http://boj.kr/f1368fdd6e164147995af83fa871f74b>

### 3 Type Printer

출력할 순서를 잘 정해주면, 그 다음은 단순 구현 문제다.

일단 Trie를 만들자. 명령 개수를 최소화하는 것은, Trie에서 문자열의 마지막 글자를 나타내는 정점을 순회하면서, Trie 상에서의 이동 거리를 최소화하는 것과 동치이다.

트리 구조이므로 재귀적으로 생각할 수 있다. 현재 정점  $v$ 에서 내려갈 수 있는 서브 트리  $T_1, T_2, \dots, T_k$ 가 있을 때 서브 트리는 재귀적으로 처리하고,  $v$ 에서 서브 트리를 방문할 순서  $P_1, P_2, \dots, P_k$ 를 정하자.

정점 방문 과정을 생각해보면,  $v$ 에서  $T_{P_1}$ 에 들어갔다가 나오고,  $T_{P_2}$ 에 들어갔다가 나오고,  $\dots$ ,  $T_{P_k}$ 에 들어간 다음 **종료하거나** 빠져나온다. 잘 생각해보면 서브 트리의 깊이가 가장 깊은 것을 마지막에 방문하면 된다는 것을 알 수 있다.

#### 3.1 Trie Example

**Usage:** Trie Implementation

**Time Complexity:** insert:  $O(|S_i|)$ , dfs:  $O(\sum_i(|S_i|)|\Sigma| \log |\Sigma|)$

**Author:** JusticeHui

```
struct Node{
    Node *ch[26] = {0};
    int ter, len; // terminal, length
    Node() : ter(-1), len(0) {}
    void insert(const char *key, int num, int ln){
        if(*key == 0){
            ter = num; len = max(ln, len); return;
        }
        if(!ch[*key - 'a']) ch[*key - 'a'] = new Node();
        ch[*key - 'a']->insert(key+1, num, ln);
        len = max(len, ln);
    }
};

vector<int> order;
void dfs(Node *v){
    if(v->ter != -1) order.push_back(v->ter);
    sort(v->ch, v->ch+26, [&](Node *a, Node *b){
        if(a == 0) return false;
        if(b == 0) return true;
        return a->len < b->len;
    });
    for(int i=0; i<26; i++) if(v->ch[i]) dfs(v->ch[i]);
}
```

<http://boj.kr/d9bb831cc7d14ceb9900b64a5ef3a9f4>

## 4 금광

직사각형의 각 변에 하나 이상의 금광이 걸쳐 있는 경우만 봐도 정답을 찾을 수 있다.  $x, y$  좌표를 각각 압축해주면 서로 다른  $x, y$  좌표 값이 최대  $2N \leq 6000$  개씩 존재한다.

Naive한 풀이를 먼저 생각해보자.

직사각형의 가로 변 2개를 선택하는 경우는  $O(N^2)$  가지이다. 가로 변 2개를 고정하면 최대 부분합 문제로 바뀌게 되고, 각 경우에 대해  $O(N)$ 에 문제를 풀 수 있다. 가로 변을 선택하는 것은 필수적이므로 최대 부분합 문제를 빠르게 풀 방법을 생각해야 한다.

최대 부분합 문제는 분할 정복을 이용해  $O(N \log N)$ 에 해결할 수 있다.

세그먼트 트리는 분할 정복을 메모이제이션하는 자료구조이다.

두 개를 합치자.

세그먼트 트리로 최대 부분합 문제를 푸는 것은 어렵지 않다.

각 정점마다 (1) 구간의 왼쪽 끝 점을 포함하는 최대 부분합 (2) 구간의 오른쪽 끝 점을 포함하는 최대 부분합 (3) 구간의 전체 합 (4) 구간 내에서 최대 부분합을 저장하고 있으면 최대 부분합 문제를 해결할 수 있다.

정해는 다음과 같다.

일단 점을  $y$  좌표 기준으로 정렬한다. 그 다음 직사각형의 가로 변 하나( $= y_1$ )을 고정하자.

$y$  좌표가  $y_1$  이상인 점들을 순서대로 세그먼트 트리에 넣어주면서, 동시에 최대 부분합을 구해 최댓값을 갱신하면 된다.

$y$  좌표가 같은 점들은 한 번에 넣어야 한다는 것을 주의하자.

세그먼트 트리에  $O(N^2)$ 번 쿼리를 날리므로  $O(N^2 \log N)$ 에 문제를 풀 수 있다.

<http://boj.kr/413fea0bf09e44e9bb703f33df8996ef>

## 5 장애물 경기

장애물을 x좌표 순으로 정렬하자.

장애물을 만날 때마다 (y좌표, 이동 거리)를 pair로 관리해주면 된다. 가로 방향 이동 거리는 항상 일정하기 때문에, 세로 방향 이동 길이만 관리하면 된다.

장애물을 만날 때마다 위/아래로 분기되기 때문에, 단순하게 정보를 관리해주면  $2^N$ 개의 pair를 관리해야 하므로 문제를 풀 수 없다.

y좌표 별로 이동 거리가 짧은 것만 들고 있으면, 동시에 최대  $2N$ 개의 pair만 관리하면 된다.

pair는 set으로 관리하면 된다. 장애물을 만날 때마다, set을 순회하면서 각 장애물이 커버하는 범위에 대해 최단 거리를 구한 뒤, 위/아래로 가는 정보를 갱신해주면 된다.

<http://boj.kr/306165785ec34d2a94a6293d37c666b9>

## 6 Favorite Colors

a, b가 동일한 소 c를 admire한다면 a와 b는 무조건 같은 색깔이다.  $\rightarrow$  a와 b를 똑같이 취급해도 된다.

$G_a = \mathbf{a}$ 를 admire하는 소들의 리스트라고 하면, a와 b가 같은 소를 admire할 때  $G_a$ 와  $G_b$ 를 병합해도 된다.

더 이상 병합할 수 없을 때까지 계속 합쳐주고, 마지막에 각 집합의 색깔을 차례대로 배정해주면 된다.

리스트를 합칠 때는 작은 리스트에 있는 정보를 큰 리스트로 옮기면 각 원소가 최대  $O(\log N)$ 번만 이동하므로 효율적으로 리스트를 병합할 수 있다.

<http://boj.kr/6ca68fcf20724b8fb513480c58adcaf0>

## 7 Springboards

오른쪽/위로만 이동한다? DP 문제라는 것을 유추할 수 있다.

각 스프링보드의 시작점까지 최단 거리만 알면 된다.

$D_i$ 를  $i$ 번째 스프링보드의 시작점까지 가는 거리로 정의하고 상태 전이를 생각해보자.

- $(0, 0)$ 부터 걸어가면  $X1_i + Y1_i$
- 적당한  $j$ 에 대해,  $j$ 번째 스프링 보드를 거쳐서 가면  $D_j + (X1_i - X2_j) + (Y1_i - Y2_j)$ 
  - 약간 변형하면  $(X1_i + Y1_i) + (D_j - X2_j - Y2_j)$
- **최솟값을 취하면 된다.**

현재  $i$ 번째 스프링보드를 보고 있다면,  $D_i$ 는  $X2_j \leq X1_i \cap Y2_j \leq Y1_i$ 인 모든  $j$ 에 대해  $(X1_i + Y1_i) + (D_j - X2_j - Y2_j)$ 의 최솟값이 된다.

시간점과 끝점을 x, y좌표 오름차순으로 정렬해주면,  $j < i$ 일 때  $x_j > x_i \cap y_j > y_i$ 인 경우가 없다.  $i$ 번째 점을 보고 있을 때,  $D_i$ 를 계산하기 위해 필요한 모든  $D_j$ 가 이미 계산되었다는 것을 의미한다.

스위칭을 하자.

현재 보고 있는 점이 스프링보드의 도착점이라면  $D_j - X2_j - Y2_j$ 를 업데이트하고, 시작점이라면  $D_i$ 를  $(X1_i + Y1_i) + (D_j - X2_j - Y2_j)$ 의 최솟값으로 갱신해야 한다.

점들의 x좌표는 이미 단조증가하므로 y좌표만 신경써도 된다. 구간의 최솟값을 관리하는 세그먼트 트리를 이용해 문제를 해결할 수 있다.

좌표 범위가 10억이므로 좌표 압축을 하거나, 다이나믹 세그먼트 트리를 구현하자.

<http://boj.kr/1ddd3eb0c6f2490fbae7c71b416889f0>

## 8 Wall Clocks

각 사람의 시야를 구간으로 표현할 수 있다. 원형으로 문제를 풀기 전에 선형을 먼저 생각해보자.

끝점 기준으로 정렬한 다음, 최대한 뒤에 시계를 배치하는 그리디로 문제를 풀 수 있다.

시간 복잡도는  $O(N \log N)$ 이다.

이 문제는 원형이긴 한데  $N \leq 1000$ 이다. 선형일 때의  $O(N \log N)$  풀이를  $N$ 번 반복하면 될 것 같다.

구간 하나를 잡아서, 그 구간의 시작점을 끊어주면 선형으로 바뀐다.  $N$ 번의 시도에서 시계를 가장 적게 사용할 때 사용하는 시계의 개수를 출력하면 된다.

<http://boj.kr/02dfedce337c456381cc162eb7f28a78>

## 9 최단 경로와 쿼리

$N$ 이 매우 작은 점을 이용해 풀이를 생각하자.

$L \sim R$ 번째 열 안에 있는 쿼리를 처리하는 것을 생각해보자.

$L < M < R$ 인 적당한  $M$ 을 잡았을 때, 쿼리의 시작점과 끝점이  $M$ 번째 열을 기준으로 서로 반대편에 있다면 **최단 경로는 항상  $M$ 번째 열을 지난다.**

각 열에는  $N$  ( $\leq 5$ ) 개의 칸이 있으므로, Dijkstra Algorithm을  $N$  번 돌리면  $M$ 번째 열을 지난 모든 경로의 최단 거리를 알 수 있다.

이제  $M$ 번째 열을 지난지 않는 쿼리를 처리해야 한다.

최단 경로가  $M$ 번째 열을 지난지 않는다는 것은, 쿼리의 시작점과 끝점이 한쪽에 쏠려있다는 것을 의미한다.  $[L, M-1]$ ,  $[M+1, R]$  구간에 대해 각각 분할 정복을 하면 된다.

$M$ 을 구간의 중점 ( $= \lfloor \frac{L+R}{2} \rfloor$ )으로 잡으면,  $O(NM \log NM)$  째 Dijkstra Algorithm을  $N$  번 돌리는 과정을  $O(\log M)$  번 반복한다.

$O(N^2 M \log NM \log M)$ 에 문제를 풀 수 있다.

<http://boj.kr/400f2810d42c42c6bd374fa45d5e6469>

## 10 케이크 3

일단 어떻게  $M$ 개를 잘 선택했다고 하자.  $\sum V_i$ 는 일정하기 때문에,  $M$ 개를 잘 배치해서 인접한  $C_i$ 의 차이를 최소화하면 된다.

잘 생각해보면  $C_i$  오름차순으로 정렬하는 것이 최적임을 알 수 있고, 이때 인접한 원소 차이의 합은  $2(\max(C_i) - \min(C_i))$ 이다. 모든  $C_i$ 에 2를 곱해주면  $\max(C_i) - \min(C_i)$ 가 된다.

$M$ 개를 잘 선택해서  $\sum V_i - \max C_i + \min C_i$ 를 최대화하는 문제를 풀면 된다.

$O(N^3)$  풀이를 먼저 알아보자.

선택할  $C_i$ 의 최대/최소를 고정시키고, 선택 가능한 원소 중  $V_i$ 가 가장 큰  $M$ 개를 선택하면 된다.

구현 방식에 따라  $O(N^3) \sim O(N^3 \log N)$  정도 걸리고, 실제 대회에서 5점을 받을 수 있다.

$O(N^3)$  풀이를 조금 변형하면  $O(N^2 \log N)$  풀이를 얻을 수 있다.

각 조각을  $(C_i, V_i)$  pair 순으로 정렬하자. 정렬한 배열에서  $i < j$ 인  $i, j$ 를 선택하는 것은  $O(N^3)$  풀이에서  $C_i$ 의 최대/최소를 선택하는 것과 똑같다.

시작점  $i$ 를 정한 뒤  $j$ 를 하나씩 증가시키면서, multiset이나 priority-queue로  $V_i$ 가 가장 큰  $M$ 개를 관리하면 된다.

$O(N^2)$  개의 구간에 대해 각각 amortized  $O(\log N)$  시간이 걸리므로  $O(N^2 \log N)$ 이며, 실제 대회에서 24점을 받을 수 있다.

만점 풀이를 알아보자.

앞에서 소개한 풀이처럼  $O(N^2)$ 개의 구간을 모두 보는 것으로는  $O(N^2)$ 보다 빠르게 해결할 수 없다. 고려하는 구간의 개수를 줄여야 한다.

$O(N^2 \log N)$  풀이를 다시 보자.

시작점  $i$ 를 고정했을 때 정답을 얻을 수 있는 가장 작은  $j$ 를  $P_i$ 라고 하면,  $P_i \leq P_{i+1}$ 이 성립한다는 것을 직관적으로 알 수 있다.

최적 위치가 단조증가하므로 Divide and Conquer Optimzation을 적용할 수 있다.

원소들이  $C_i$  기준으로 정렬되어 있으므로  $i \in [s, e]$ 인  $i$ 에 대해,  $V_i$  중 가장 큰  $M$ 개의 합을 구하는 연산을 빠르게 할 수 있으면 된다.

이 연산은 Persistent Segment Tree를 이용해  $O(\log N)$ 에 할 수 있다.

$T(N) = 2T(N/2) + O(N \log N)$ 이므로  $O(N \log^2 N)$ 에 문제를 해결할 수 있다.

<http://boj.kr/17a7f2763c0b415bbe39201b2143c4bc>

## 11 벽 칠하기

$\sum f(k)^2 \leq 400\,000$ 이므로  $f(k) \leq \sqrt{400\,000}$ 이다.

$T(i, j)$ 를  $i$ 번째 구간을  $j$ 번째 일꾼으로 칠할 수 있으면 1, 칠할 수 없으면 0으로 정의하자. 이때  $T$ 에서 1의 개수는 최대  $N\sqrt{400\,000}$ 이다.

$T$ 를 원통처럼 둥글게 말았을 때, 1로 구성된 길이  $M$  이상인 우하향 대각선이 존재한다면 그 위치에 지령을 내릴 수 있다.

$D(i, j)$ 를  $(i, j)$ 에서 끝나는 우하향 대각선의 최대 길이라고 하자.

토글링을 하면 DP 배열을 시간 복잡도  $O(N\sqrt{400\,000})$ , 공간 복잡도  $O(M)$ 에 채울 수 있다.

$D(i, *) \geq M$ 이라면 지령을 통해 구간  $[i - M + 1, i]$ 를 칠할 수 있다. 칠할 수 있는 구간을 모두 모은 다음 그리디하게 색칠하면 된다.

<http://boj.kr/b16ffeae4cd14efa8034e1568b26f1aa>

## 12 Xtreme NP-Hard Problem

$N < K$ 이거나  $M < K$ 이면 간선  $K$ 개로 이루어진 경로가 없다. 그러므로  $\min(N, M, K) \leq X$ 라는 조건은  $K \leq X$ 라고 생각할 수 있다.

$K = 1, 2, 3, 4, 5$ 일 때의 풀이를 각각 알아보자.

$K = 1$ 인 경우에는 1과  $N$ 을 연결하는 간선이 있는지 확인하고, 있다면 그 간선의 가중치를 출력하면 된다.

$K = 2$ 인 경우에는 1과  $N$ 에서 동시에 갈 수 있는 정점  $X$ 에 대해,  $d(1, X) + d(X, N)$ 의 최솟값을 구하면 된다.

$K = 3$ 인 경우에는 경로가  $1 \rightarrow x \rightarrow y \rightarrow N$  형태다.

$A_x$ 는 1과  $x$ 를 연결하는 간선의 가중치,  $B_y$ 는  $y$ 와  $N$ 을 연결하는 간선의 가중치로 정의하자. 간선이 없는 경우에는  $\infty$ 이다.

모든 간선  $(x, y)$ 를 보면서,  $A_x + d(x, y) + B_y$ 의 최솟값을 보면 된다.

$K = 4$ 인 경우에는 경로가  $1 \rightarrow x \rightarrow y \rightarrow z \rightarrow N$  형태다.

1초과  $N$  미만인 모든 자연수  $y$ 를 보면서,  $y$ 와 연결된 두 정점  $x, z$ 에 대해 ( $x \neq y$ )  $(A_x + d(x, y)) + (d(y, z) + B_z)$ 의 최솟값을 구하면 된다.

$(A_x + d(x, y))$  중 가장 작은 2개,  $(d(y, z) + B_z)$  중 가장 작은 2개를 구해서,  $x \neq z$ 이면 최솟값을 갱신해주면 된다.

$K = 5$ 인 경우에는 경로가  $1 \rightarrow w \rightarrow x \rightarrow y \rightarrow z \rightarrow N$  형태다.

$A'_x$ 를  $1 \rightarrow w \rightarrow x$  경로 중 가중치가 가장 작은 3개,  $B'_y$ 를  $y \rightarrow z \rightarrow N$  경로 중 가중치가 가장 작은 3개라고 하자. 이때  $A'$ 에서는 (거리,  $w$ )를,  $B'$ 에서는 (거리,  $z$ )를 저장해야 한다.

모든 간선  $(x, y)$ 를 보면서,  $A'_x + d(x, y) + B'_y$ 의 최솟값을 구하면 된다. 단,  $w, x, y, z$ 가 모두 다 다른 경우에만 갱신을 해야 한다.

<http://boj.kr/3ec887f36b5a43229d81d80ffbf68088>

## 13 Remote Control

모든 좌표(쿼리)를 입력 받은 다음 한 번에 처리하자.

각 턴마다 차  $Q$ 대를 전부 움직이는 것은 절대 안 될 것 같으니, 벽을 움직일 것이다.

만약 어떤 차가 벽과 충돌하면 그 행동은 캔슬된다. 이런 경우, 그 차도 벽이 이동하는 방향으로 같이 밀어주면 된다.

어떤 두 차가 동일한 칸에 속한다면, 그 시점 이후에도 계속 같은 칸에 속하게 된다.  
Union-Find로 묶어서 관리하면 된다.

<http://boj.kr/4b01d5a35b364a3c8b026c438941be19>

## 14 성벽

성벽을 (1) 북서쪽에서 뻗어나가는 성벽과 (2) 남동쪽에서 뻗어나가는 성벽으로 나눠서 생각할 것이다. 북서-동남을 연결하는 대각선은  $H + W - 1$ 개가 있고, 각 대각선에 대해 각각 정답을 구하면 된다.

각 점에서 4방향으로 뻗어나갈 수 있는 최대 길이를 각각 구하자.  $(E, W, S, N)$  DP를 이용하면  $O(HW)$ 에 구할 수 있다.

각 대각선의 윗부분부터 쭉 훑으면서, 현재 점  $(i, j)$ 에서 뻗어나가는 북서쪽 성벽과 매칭 가능한 남동쪽 성벽의 개수를 구하는 방식으로 답을 구할 것이다. Inversion Counting과 유사한 방법으로 구할 수 있다.

어떤 대각선에 대해...

일단 대각선 원에 있는 모든 점  $(i, j)$ 에 대해, 그 점에서 시작하는 남동쪽 성벽이 어디까지 뻗어나갈 수 있는지 구하자.  $\rightarrow i - \min(W(i, j), N(i, j)) + 1$   
 $\{i - \min(W(i, j), N(i, j)) + 1, i\}$  pair를 저장한 배열  $v$ 를 정렬하자.

정렬된 배열  $v$ 를 쭉 보면서, Inversion Counting 느낌으로 답을 구하면 된다. 자세한 구현은 아래 코드를 참고.

### 14.1 Solve like Inversion Counting

**Usage:** solve this problem

**Time Complexity:**  $O((H + W) \log(H + W))$

**Author:** JusticeHui

```
for(int i=si, j=sj; i<=n && j<=m; i++, j++){
    range.emplace_back(i-min(north[i][j], west[i][j])+1, i);
    sort(all(range));

    for(int i=si, j=sj; i<=n && j<=m; i++, j++){
        while(pv < range.size() && range[pv].x <= i) update(range[pv++].y, 1);
        ans += query(i+1-1, i+min(south[i][j], east[i][j])-1);
    }
}
```

<http://boj.kr/8b5b2406ed854ac49d49d6e2b8da0238>

## 15 NP-Hard

정답은 감소하다 1을 찍고 증가하는 형태가 될 것이다. 1부터 시작해서 양 끝에 수를 적절히 붙여나가면 된다.

$D(l, r) =$  왼쪽 끝에  $l$ 이 있고, 오른쪽 끝에  $r$ 이 있을 때 최소 시간으로 정의하자.  
상태 전이는 왼쪽 또는 오른쪽에  $\max(l, r) + 1$ 을 붙이는 것이다.

상태 공간은  $O(N^2)$ 개이고, 각 상태마다  $O(1)$ 에 답을 구할 수 있으므로  $O(N^2)$ 에 문제를 해결할 수 있다.

<http://boj.kr/eb844e7b477845a186fccd7a1e0130d5>

## 16 Namje Adventure

현재 줄의 상태를  $L$ 개의 비트로 표현할 수 있다.

현재 상태  $now$ 에서 맨 위에 있는 사람이 이동해서 상태를  $nxt$ 로 바꾸는데 필요 한 에너지를  $W$ 라고 하자.  $now$ 에서  $nxt$ 로 가는 가중치가  $W$ 인 간선을 만들어서 연결하자.

깊이가 1인 곳부터  $N$ 인 곳까지 사람이 있는 상태에서, 깊이가  $D - N - 1$ 인 곳부터  $D$ 인 곳까지 사람이 있는 상태로 이동하는 최소 비용을 구하면 된다. 이는 (맨 위  $N$ 개의 칸에 사람이 있는 상태)에서 출발해 (간선  $D - N$ 개)를 거친 뒤, 다시 (맨 위  $N$ 개의 칸에 사람이 있는 상태)로 돌아오는 것이다.

간선  $D - N$ 개를 거쳐서 가는 최소 비용은 인접 행렬을  $D - N$ 제곱하는 것으로 구할 수 있다.

단, 이때 행렬 곱 연산은 Floyd-Warshall Algorithm으로 정의한다.

$L \leq 12$ 라서 정점이  $2^{12} = 4096$ 개라고 생각할 수 있지만, 실제로 가능한 상태는  $\binom{N}{L}$ 가지 밖에 없다. 좌표 압축을 하면 최대 220개의 정점만 봐도 된다.

<http://boj.kr/e992b4f61faf4ef3b0c2d5c7ccbc1c91>

## 17 타일 놓기

타일의 개수를 최소화하는 것은, 모서리를 최대한 많이 끊어낸다는 것으로 생각할 수 있다. 이때 정답은 (타일을 놓아야 하는 칸의 개수) - (끊은 모서리의 개수)가 된다.

타일은  $1 \cdot K$  꼴이기 때문에 각 둉어리가 꺾이면 안 된다. 그러므로 L 모양이 되도록 모서리를 끊을 수 없다.

격자의 각 모서리를 정점으로 잡고, 끊으면 L 모양이 나오는 두 정점을 간선으로 연결하자. 선택한 정점이 서로 인접하지 않도록, 최대한 많은 정점을 선택하면 된다.

### 최대 독립 집합

만들어진 그래프는 이분 그래프이므로, 이분 매칭을 통해 최대 독립 집합의 크기를 얻을 수 있다.

Dinic's Algorithm이나 Hopcroft-Karp Algorithm을 이용하면  $O(NM\sqrt{NM})$ 에 문제를 해결할 수 있다.

<http://boj.kr/2a5e0503718549058cd338d378868a58>

## 18 테이터 제작

문제를 요약하면 다음과 같다.

- 정점이  $N$ 개, 간선이  $M$ 개, 면이  $K$ 개인 평면 그래프 제작 (outer face 제외)
- self loop, multi edge 없음
- (중요) 좌표 범위 제한 79

좌표 범위 제한만 없으면 아주 쉬운 문제인데, 좌표 제한 때문에 어려운 문제가 되었다.

$V, E, F, C$ 를 각각 그래프의 정점, 간선, 면, 컴포넌트의 개수라고 하면 평면 그래프에서는  $V - E + F = C + 1$ 이 성립한다. 문제에서 주어진  $N, M, K$ 는 각각  $N = V, M = E, K = F$ 이므로  $N - M + K = C$ 이다.

또한 self loop와 multi edge가 없는 평면 그래프에서는  $M \leq 3N - 6$ 이 성립한다. 그러므로  $N$ 개의 정점과  $3N - 6$ 개의 간선을 좁은 공간 안에 다 밀어넣는 방법을 찾은 뒤,  $M < 3N - 6$ 이라면 간선을 제거해주면 된다.

**Remark.**  $C \neq 1$ 인 경우에는 정점  $C - 1$ 개를 따로 빼내면  $C = 1$ 인 상황으로 만들 수 있다.

좌표 범위를 신경쓰지 않고,  $3N - 6$ 개의 간선을 만드는 방법은 간단하다.

충분히 큰 삼각형을 만든 뒤, 삼각형 내부에 점 하나를 찍고 기존 삼각형과 간선 3개로 연결하는 것을 반복하면 된다.

좁은 공간에 잘 밀어넣는 방법을 알아보자.

세 꼭짓점의 좌표가  $(1, 79), (1, 1), (79, 2)$ 인 삼각형을 만들자.

삼각형 내부에

- $y = 78$ 인 점 1개
- $y = 77$ 인 점 2개
- ...
- $y = i$ 인 점  $(79 - i)$ 개

를 넣을 수 있다. 총  $3 + \frac{78 \times 79}{2} = 3084$ 의 점을 만들 수 있고, 이는 3000개의 점을 넣기에 충분하다.

$3N - 6$ 개의 간선을 이용해 삼각형을 조개는 것은 Figure 1처럼 하면 된다.

빨간 간선은 정점들이 연결 그래프를 이루기 위해서 꼭 필요한 간선이고 파란색 간선은  $M$ 의 값에 따라 제거해도 되는 간선이다.

$M$ 값에 따라 파란색 간선을 적절히 추가/제거해서 원하는 그래프를 만들 수 있다.  $C \neq 1$ 인 경우에는 남은 정점  $C - 1$ 개를 큰 삼각형 영역 바깥쪽에 배치하면 된다.

<http://boj.kr/42ecf90ef0c5458e823b76608f3e492e>

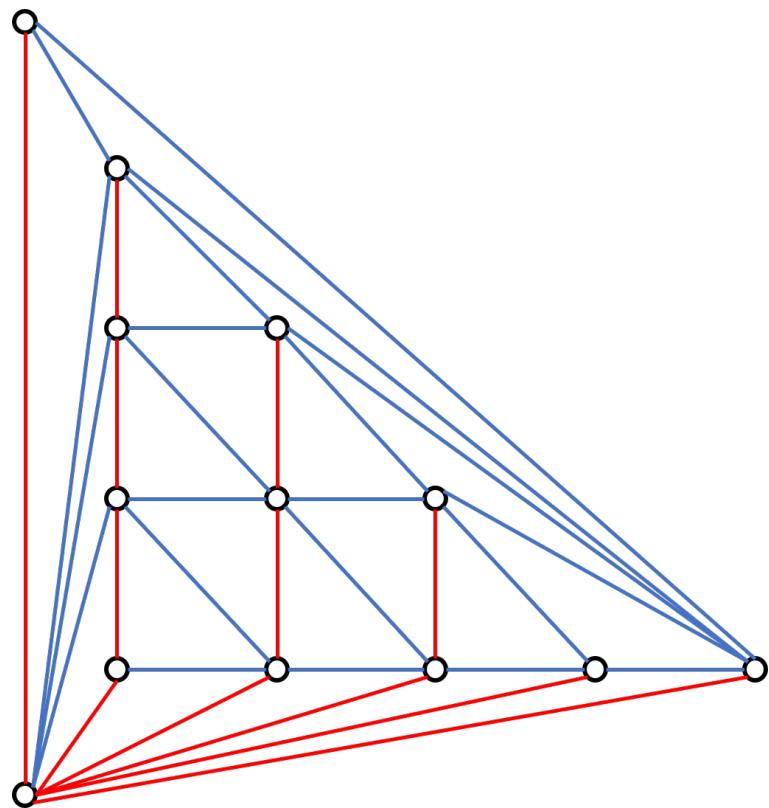


Figure 1: ucpc-data-making-solution

## 19 부분 문자열의 개수

만든 문자열의  $i$ 번째 글자가  $c$ 이면서, 패턴(주어진 문자열)의  $j$ 번째 글자까지 매칭되는 경우의 수를  $D(i, j, c)$ 로 정의하자.

만든 문자열의  $i$ 번째 글자와 패턴의  $j$ 번째 글자가 매칭됐는데  $i + 1$ 과  $j + 1$ 이 매칭되지 않는다면? KMP의 Failure Function을 따라가면 된다.

KMP + DP로 문제를 풀 수 있다.

<http://boj.kr/b1c69aab4ff047f79feeffdd5f306735>

## 20 Mountains

(오른쪽에 있는  $i$ 가 왼쪽에 있는  $j$ 를 볼 수 있는 것)은 ( $j < k < i$ 인 모든  $k$ 에 대해  $ccw(j, k, i) \geq 0$ )과 동치이다. 이해가 안 된다면 그림을 그려보자.

$D(j, i)$ 를  $[j, i]$  구간의 답이라고 정의하자. Naive하게 계산하면  $O(N^3)$ 이고 당연히 TLE를 받게 된다.

$i$ 번째에서 볼 수 있는 곳을 생각해보자. 볼 수 있는 곳을 기준으로 **분할된** 구간에 대해, 각각 독립적으로 문제를 풀고 합쳐줘도 된다.

$i$ 를 고정하고  $j$ 를  $i - 1$ 부터 왼쪽으로 이동하면서 DP를 계산한다.

구간  $(j, i]$ 에서  $i$  볼 수 있는 가장 왼쪽 사람  $left4$ 를 관리하자.  $ccw(j, left, i) \geq 0$ 이라면  $i$ 는  $j$ 를 볼 수 있으며,  $(j, left)$  사이에 있는 사람들은  $left$ 에 가려져서 안 보인다.

이 정보를 이용해  $ccw(j, left, i) \geq 0$ 을 만족하는  $j$ 가 나올 때마다  $D(j+1, left-1)$ 을 더해가면서 문제를 풀 수 있다.

각  $i$ 에 대해  $left$ 는 최대  $i$ 번 움직이므로  $O(N^2)$ 에 문제를 해결할 수 있다.

<http://boj.kr/f69b200d53d447b0817ad0b99f0d090a>

## 21 소방차

매칭하는 것을 구간이라고 생각하자.

정답이 어떤 형태인지, 혹은 특정한 형태로 강제할 수 있는지 생각해보자.

**Lemma 1.** 구간들이 서로를 완전히 포함하거나 분리된 형태의 최적해가 존재한다.

Proof.  $a \leq b \leq c \leq d$ 일 때  $(a, c), (b, d)$ 를 매칭하는 것이 최적이라고 하자.  $(a, d), (b, c)$ 로 바꿔도 거리는 동일하다.

**Lemma 2.** 최적해에  $[l, r]$  매칭이 존재한다면, 이 구간 안에 있는 소방차와 펌프는 모두 매칭되어 있다.

Proof. 그렇지 않으면 적당히 바꿔서 거리를 더 줄일 수 있다.

구간의 포함 관계를 트리(혹은 포레스트)처럼 생각할 수 있다. 이때 같은 depth에서는 모든 구간이 분리되어 있다.

입력으로 들어온 모든 좌표를 정렬하고, 펌프가 나오면 현재 depth에 넣고 depth++, 소방차가 나오면 -depth하고 그 depth에 넣자.

같은 depth에서는 펌프와 소방차가 번갈아가며 나오게 된다. 원소가 짹수 개라면 순서대로 매칭하면 되고, 홀수 개라면 어떤 하나를 빼고 매칭하는 경우를 모두 확인해야 한다. 이 작업은 선형 시간에 할 수 있다.

<http://boj.kr/91b9e89374e34559b37f2477f06e1f89>

## 22 편식

올리브의 Convex Hull 위에 있는 점만 고려해도 된다.

피자의 두 꼭짓점을 잇는 선분이 볼록 껍질과 닿지 않으면 되는데, 어떻게 빠르게 판별할 수 있을까?  $O(N^2)$ 개의 선분을 다 보는 것은 안 될 것 같으니 다른 방법을 생각해보자.

필요한 개념

- 볼록 껍질 (Monotone Chain)
- 삼분 탐색
- 투 포인터
- 볼록 다각형의 접선
- 사선 공식

피자의 두 꼭짓점을 잇는 선분과 평행하면서, 올리브 볼록 껍질에 접하는 두 직선을 그려보자. 빨간색 선분이 두 직선 사이에 있다면 불가능하고, 그렇지 않으면 가능하다. 기울기가 주어졌을 때 볼록 다각형의 접선을 구하는 것은, 볼록 다각형을 윗 껍질과 아랫 껍질로 분리한 뒤 삼분 탐색을 하면  $O(\log M)$ 에 구할 수 있다. 어떤 선분이 두 직선 사이에 있다는 것은, (한 직선의 y절편)  $\leq$  (선분의 y절편)  $\leq$  (다른 직선의 y절편)이라는 것을 의미한다.

어떤 선분으로 자를 수 있는지 판별하는 것은  $O(\log M)$ 에 할 수 있다.

잘 생각해보면,  $O(N^2)$ 개의 선분을 모두 보지 않아도 된다.

투 포인터로 선분의 양 끝 점을 관리하면 된다.

$i$ 번째 점과  $i+1, i+2, i+3, \dots, j$ 번째 점을 연결할 수 있다고 하자. 그러면  $i+1$ 번째 점은  $i+2, i+3, \dots, j$ 번째 점과 연결할 수 있고,  $i+2 \sim j-1$ 번째 점과 연결하는 것은 최대 넓이가 될 수 없다. 그러므로  $i+1$ 번째 점은  $j$ 번째 점부터 확인하면 된다.

$O(N)$ 개의 선분만 확인하면 되고, 이때 넓이는 신발끈 공식을 이용하면 상수 시간에 관리할 수 있으므로  $O(N \log M)$ 에 문제를 풀 수 있다.

<http://boj.kr/2626bd351a2941c787e3c8b7e1d1de0e>

## 23 Stretch Rope

배낭의 무게가  $[A_i, B_i]$ 라는 구간으로 표현되는 0/1 Knapsack Problem이다.  $D(i, j)$ 를  $i$ 번째 물건까지 고려했을 때 무게 합이  $j$ 인 경우의 최소 가치라고 하자. 상태 전이를 생각해보자. 무게 합이  $j$ 가 되기 위해서는 기존 무게가  $[j - B_i, j - A_i]$  범위 안에 있어야 한다. 구간의 최솟값을 deque 또는 segment tree로 구해주면 문제를 해결할 수 있다.

<http://boj.kr/4934b0abd6f84a53a78097507d54b7e5>

## 24 Alternating Current

어떤 선분  $a$ 가  $b$ 에 완전히 포함된다면  $a$ 와  $b$ 에는 서로 다른 색을 배정하면 된다.  
어떠한 선분에도 완전히 포함되지 않는 선분들만 보면서 색깔을 잘 배정하고, 그 선분에 포함되는 선분에는 반대의 색을 배정하는 방식으로 문제를 풀자.  
다른 선분에 의해 완전히 포함되는 선분을 제거하는 것은 Sliding Window 기법을 이용해 구현할 수 있다. 아래 코드 참고

### 24.1 Sliding Window

Usage: get eliminated segments

Time Complexity:  $O(M \log M)$

Author: JusticeHui

```
struct Info{
    int s, e, x, eli, color;
    Info() : Info(0, 0, 0) {}
    Info(int s, int e, int x)
        : s(s), e(e), x(x), eli(-1), color(-1) {}
    bool operator < (const Info &t) const {
        return make_pair(s, -e) < make_pair(t.s, -t.e);
    }
};
void calc_elimination(){
    deque<Info> dq;
    sort(a+1, a+m+1);
    for(int i=1; i<=m; i++){
        if(dq.empty() || dq.back().e < a[i].e) dq.push_back(a[i]);
        else tmp[a[i].x] = dq.back().x;
    }
    while(dq.size() && dq.front().e <= ed)
        tmp[dq[0].x] = ed_idx, dq.pop_front();

    for(auto i : dq) v.push_back(i);
    for(int i=1; i<=m; i++) if(tmp[a[i].x]) a[i].eli = tmp[a[i].x];

    iota(par, par+101010, 0);
    for(int i=1; i<=m; i++) if(a[i].eli != -1) merge(a[i].eli, a[i].x);
    for(int i=1; i<=m; i++){
        if(a[i].eli != -1){
            a[i].eli = find(a[i].x);
            g[a[i].eli].push_back(a[i].x);
        }
        else g[a[i].x].push_back(a[i].x);
    }
}
```

```
    }  
}
```

포함되는 선분을 제거하자. 각 구간은 최소 한 개 이상의 선분으로 덮여 있다. 제거되지 않은 선분을 정렬해서 보면, 계단 형태로 배치된 것을 알 수 있다.  
만약 정답이 존재한다면, 선분을 정렬했을 때 색을 교차해서 배정했을 때 정답이 무조건 존재한다.

제거되지 않은 선분의 개수를  $K$ 라고 하자.  $K$ 가 짝수라면 단순히 색깔을 번갈아 가며 배치한 뒤, 정답이 될 수 있는지 확인하면 된다.  
 $K$ 가 홀수라면 아래  $K$ 가지 경우를 모두 확인해야 한다.

1. 01010 (1번 선분부터 시작)
2. 11010 (2번 선분부터 시작)
3. 10010 (3번 선분부터 시작)
4. 10110 (4번 선분부터 시작)
5. 10100 (5번 선분부터 시작)

$K$ 가지 경우를 Naive하게 모두 확인하면 서브태스크 1, 2, 3을 해결해 55점을 받게 된다.

위에서 나열한 5가지를 잘 보면,  $i$ 번째 비트열과  $i + 1$ 번째 비트열은  $i$ 번째 비트만 다르다. 그러므로  $K$ 가지 경우를 모두 확인할 때 각 선분의 색을 한 번만 바꿔도 된다.

Segment Tree + Lazy Propagation으로 구간의 최솟값을 관리하면 선분이 잘 덮여 있는지 빠르게 확인할 수 있다.  
그러므로  $O(M \log M + M \log N)$ 에 문제를 풀 수 있다.

<http://boj.kr/46aeae8030df4a38b8b9af87cd8b0fc9>

## 25 야바위

각 쿼리마다 구슬의 최종 위치만 알면 된다.

오프라인으로 문제를 해결하자.  $S_i$ 번째 캡(std::set)에  $i$ 번째 구슬을 넣고, 구슬을 이동시킬 것이다.

윤이가 본  $N - 1$ 개의 동작은 지문에 나와있는대로 처리하면 된다. 이후 주어진  $M$  개의 동작은 해당하는 구슬만 이동하면 된다.

두 set을 합칠 때 Small to Large를 사용하면, 각 원소는 최대  $O(\log M)$ 번 이동하게 된다.

쿼리를 정렬하는데  $O((N + M) \log(N + M))$ , 쿼리를 처리하는데  $O(M \log^2 M)$  만큼 걸리므로 시간 내에 문제를 해결할 수 있다.

<http://boj.kr/3bb8ca848aa5433e88a7d575115dbbca>

## 26 Confuzzle

Centroid Decomposition을 하면서, 각 색깔마다 현재 Centroid와 가장 가까운 정점 2개를 저장하면 된다.

<http://boj.kr/e4db954b41c54226911477abfe018c3f>

## 27 모래시계 2

경우의 수를 직접 계산하는 것보다는 여사건을 구하는 것이 편하다.

가운데 들어갈 정점을 고정하자.

나머지 정점 4개를 선택하는 경우의 수는  $\binom{N-1}{4}$ 이다. 정점 4개를 선택하는 경우에  
서, 4개를 이용해 만들어지는 두 삼각형이 겹치는 경우의 수를 빼자.

가운데 고정한 정점을 기준으로 나머지 정점을 각도 순으로 정렬하자.

가운데 고정한 정점  $c$ 와 어떤 정점  $i$ 를 연결하는 직선의 **왼쪽**에 있는 정점의 개수를  
 $K_i$ 라고 하자.

$c, i$ , 그리고 나머지 정점 3개로 모래 시계를 만드려고 할 때, 두 삼각형이 겹치는  
경우는  $\sum \binom{K_i}{3}$  가지이다.

$K_i$ 를 구하는 것은 투 포인터를 이용해  $O(N)$ 에 구할 수 있고, 각도 정렬은  $O(N \log N)$   
이므로  $O(N^2 \log N)$ 에 문제를 풀 수 있다.

<http://boj.kr/83e1e2e60a504c259e204005a73faaca>

## 28 Salty Fish

아래 두 가지 행동을 적절히 해서 이득을 최대화하는 것이 목적이다.

1. 카메라를 매수하고 정점을 먹는 것
2. 정점을 포기하는 것

일단 모든 정점을 먹는 것을 기본으로 하고, 각 행동을 선택함으로써 손해를 보게 되는 비용을 최소화하자.

각 행동으로 인해 손해를 보게 되는 비용을 생각해보자.

1. 카메라를 매수하는 경우, 카메라의 가격인  $c_i$ 만큼 손해
2. 정점을 포기하는 경우, 그 정점에 있는 사과의 개수인  $a_i$ 만큼 손해

문제를 Min Cut으로 모델링할 수 있을 것 같다!

- Source에서 카메라로 가는 가중치가  $c_i$ 인 간선
- 카메라에서 그 카메라가 담당하는 정점으로 가는 가중치가  $\infty$ 인 간선
- 정점에서 Sink로 가는 가중치가  $a_i$ 인 간선

이렇게 그래프를 모델링하면, 정답은  $\sum a_i - \text{mincut}$  된다.  
 $\text{maxflow} = \text{mincut}$ 이므로 정답은  $\sum a_i - \text{maxflow}$ 이다.

그래프의 크기가 매우 크기 때문에( $v \leq 600,000$ ) 일반적인 플로우 알고리즘을 사용하면 안 되고, 그래프의 특성에 맞게 알고리즘을 설계해야 한다.  
기본적인 흐름은 Ford-Fulkerson처럼 증가 경로를 찾아서 유량을 흘리는 것을 따라간다.

$v$ 를 루트로 하는 서브 트리의 정점 중, 1번 정점과  $d$ 만큼 떨어져 있는 정점에서 Sink로 가는 간선들의 잔여 유량의 합을  $D(v, d)$ 로 정의합시다.  
Tree DP처럼 DFS를 하면서 아래부터 처리합니다.

- 현재 정점  $v$ 에 있는 카메라는 깊이가  $dep[v] + k_i$ 인 정점까지 관리
- 흘릴 수 있는 가장 깊은 곳부터 유량을 흘리는 것이 이득이다.
  - $D(v, *)$ 를 std::map으로 관리하면서, prev(upper\_bound)를 구해 유량을 흘리면 된다.
- 현재 정점과 부모 정점의  $D$  배열을 합치는 것은 Small to Large로

$O((N + M) \log^2(N + M))$  정도에 문제를 해결할 수 있다.

<http://boj.kr/f5e6072984bb46288ecec9a8491be7d6>

## 29 우체국 3

편의상 디스크립션에 나와있는  $V, P, L$  대신  $N, K, L$ 이라는 문제로 표기한다.

원형으로 구성된 배열을 적절히 자르는 문제다. 원형 구조를 다루는 것은 매우 귀찮기 때문에 한 곳을 끊어서 선형으로 만들자.

선형에서는 간단한 2차원 DP로 문제를 해결할 수 있다:

- $D(K, N) = 1 \dots N$ 번째 원소를  $K$ 개의 조각으로 잘랐을 때의 최소 비용
- $D(K, N) = \min(D(K-1, i) + C(i+1, N))$

구간  $[i, j]$ 의 비용  $C(i, j)$ 는 구간의 중앙값으로 가는 거리의 합이 되고, 이는 Prefix Sum을 미리 전처리해두면  $O(1)$ 에 계산할 수 있다.

그러므로 DP를 Naive하게 계산해주면  $O(KN^2)$ 이다. 끊을 수 있는 지점이  $N$ 개 있으므로 모든 경우를 다 보면  $O(KN^3)$ 이다.

$C(i, j)$ 는 사각 부등식을 만족하기 때문에 **Divide and Conquer Optimization**을 사용할 수 있다.

$O(KN^2)$  DP를  $O(KN \log N)$ 으로 최적화할 수 있으므로 전체 시간 복잡도는  $O(KN^2 \log N)$ 이 되어 문제를 해결할 수 있다.

<http://boj.kr/f234a838d12c4854a7e90906df7f3524>

## 30 3D Histogram

2차원의 경우는 매우 유명한 문제이고, Monotone Stack을 이용한 풀이와 분할 정복을 이용한 풀이가 잘 알려져 있다. Monotone Stack을 3차원에 적용하는 것은 어려울 것 같으니 분할 정복으로 접근해보자.

구간  $[s, e]$ 에 속한 히스토그램을 처리할 때, 중점  $m = \lfloor \frac{s+e}{2} \rfloor$ 을 지나는 모든 히스토그램을 처리한 뒤, 구간  $[s, m-1]$ 과  $[m+1, e]$ 에 대해 재귀적으로 처리할 것이다.

함수  $f(i, j)$ 와  $g(i, j)$ 를 아래와 같이 정의하자.

$$f(i, j) = \min_{i \leq k \leq j} (a_k)$$

$$g(i, j) = \min_{i \leq k \leq j} (b_k)$$

구간의 양 끝점  $s, e$ 와 중간 지점  $m$ 에 대해, 구간의 상태를 아래 4가지로 분류할 수 있다.

1.  $f(s, m) \leq f(m, e) \cap g(s, m) \leq g(m, e)$
2.  $f(s, m) \leq f(m, e) \cap g(s, m) \geq g(m, e)$
3.  $f(s, m) \geq f(m, e) \cap g(s, m) \leq g(m, e)$
4.  $f(s, m) \geq f(m, e) \cap g(s, m) \geq g(m, e)$

1, 2를 처리할 수 있으면, 적절히 reverse하고 swap하는 것으로 3, 4번을 똑같이 처리할 수 있다. 1, 2번 케이스에 대한 풀이를 알아보자.

1번 케이스는 투포인터를 이용해  $O(e - s + 1)$ 에 해결할 수 있다.

각 시작점  $l$ 에 대해,  $f(l, r) = f(l, m) \cap g(l, r) = g(l, m)$ 인  $r$ 의 최댓값을 찾아서 최대 부피를 갱신해주면 된다. (정답 코드에서 calc1 함수 참고)

2번 케이스는 다소 복잡하다.  $(r - l + 1)f(l, m)g(m, r)$ 을 최대화하는 것이 목적이다.

식을 조금 변형하면,  $f(l, m)g(m, r)(1 - l) + f(l, m)g(m, r)r$ 이 된다.

$f(l, m)$ 으로 묶어주면,  $f(l, m) \times (g(m, r)(1 - l) + g(m, r)r)$ 이 된다.

중괄호 안쪽의 식은  $g(m, r)$ 이 기울기,  $g(m, r)r$ 이 y절편,  $(1 - l)$ 이 변수인 일차 함수로 해석할 수 있다. 일차 함수가 여러 개 있을 때 임의의 x좌표에서 최댓값을 구하는 것은 Convex Hull Trick을 이용해서 빠르게 구할 수 있으니, 이 성질을 이용해서 풀이를 찾아보자.

먼저 1번 케이스처럼 각 시작점  $l$ 에 대해,  $f(l, r) = f(l, m) \cap g(l, r) = g(l, m)$ 이 되도록 하는  $r$ 의 구간을 전처리하자.  $O(e - s + 1)$  시간이 걸린다. 이 구간의 왼쪽 끝점과 오른쪽 끝점을 각각  $L(l), R(l)$ 이라고 하자.

각 시작점  $l$ 에 대해,  $L(l)$ 부터  $R(l)$ 번째 직선들 중에서  $(1 - l)$  시점에서의 최댓값을 구하면 된다. 이것은 Segment Tree의 각 정점에서 Convex Hull Trick을 관리하는 것으로 처리할 수 있다.

Segment Tree의 각 정점에 저장되는 직선의 기울기, 쿼리로 주어지는 x좌표인  $(1 - l)$  모두 단조성이 있기 때문에 Convex Hull Trick을 선형에 처리할 수 있다.

전체 시간 복잡도는  $T(N) = 2T(N/2) + O(N \log N)$ 이 되므로  $O(N \log^2 N)$ 에 문제를 풀 수 있다.

<http://boj.kr/be6364b253e44ef0869295481796ed19>

## 31 겹치는 선분

직선은  $y = ax + b$ 로 표현할 수 있고, 두 직선의  $a, b$ 가 모두 같은 경우에만 겹칠 수 있다.

직선을  $(a, b)$  값 별로 나눈 다음, 각각에 대해 독립적으로 문제를 해결해도 충분하다.

$(a, b)$ 가 같은 경우, 간단한 스위핑을 통해 문제를 해결할 수 있다.

<http://boj.kr/0c09704849bc4339a0961657c7197d46>

## 32 Synchronization

한글 지문은 oj.uz에서 볼 수 있다:

[https://oj.uz/problem/view/JOI13\\_synchronization](https://oj.uz/problem/view/JOI13_synchronization)

HLD + set / Euler Tour Trick + set을 이용하는 풀이가 존재하지만, 풀이를 생각하는 것과 구현이 모두 귀찮다.

약간의 사전지식이 있으면 훨씬 쉬운 풀이를 소개한다.

간선 제거 쿼리가 없는 문제를 생각해보자.

각 컴포넌트의 "정답"을 루트 정점에 저장하면서, 간선이 추가되면 컴포넌트를 Union Find를 통해 합치면 된다.

이때 각 컴포넌트의 "정답"은 단순히 컴포넌트의 크기가 된다.

간선을 끊는다면? "Link Cut Tree를 쓰면 된다!"라는 단순하고도 명쾌한 해답을 찾을 수 있다.

Union Find처럼 각 컴포넌트의 "정답"을 루트 정점에서 관리하면 된다.

하지만 간선을 끊는 쿼리가 있다면 두 컴포넌트를 합칠 때 정보의 교집합이 생길 수 있다. 이는 트리의 간선이 연결하는 정점들이 바뀌지 않기 때문에 쉽게 처리할 수 있다.

간선  $E = (u, v)$ 가 마지막으로 끊어지는 시점에서의 컴포넌트의 크기  $S(E)$ 를 알고 있다면,  $E$ 가 다시 트리에 추가될 때 해당 컴포넌트의 "정답"은  $Ans(u) + Ans(v) - S(E)$ 가 된다.

간선을 끊고, 붙이고, 루트 정점을 구하는 연산은 모두 Link Cut Tree를 이용해서 처리할 수 있다.

<http://boj.kr/27a85c1b8cd74345a4e84100b4a552f6>